2. Object Oriented Programming

- Overview
- Designing ADTs
- Declaring ADTs
- Using ADTs
- Implementing ADTs
- Conclusion

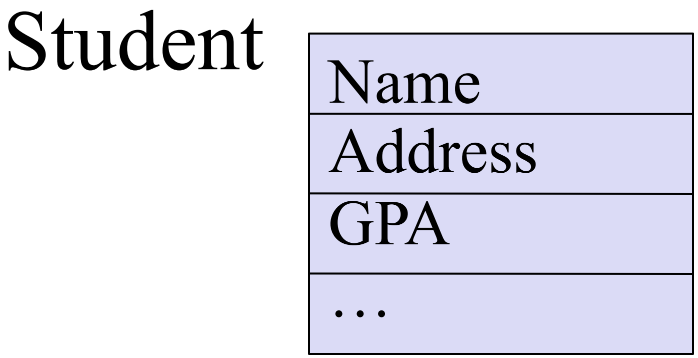
Overview

- Abstract data types (ADTs) are used to simplify program design and implementation
- The purpose of an ADT is to group data that belong together into one place
- ADTs also specify operations that make sense for that group of data
- This modular design makes it easier for several people to work together to create robust code

Designing ADTs

- Key idea is to think about nouns (data) and verbs (operations) that belong together

- Draw diagrams of data to aid in design



Student

Name

Address

GPA

…

Designing ADTs

- Make point form list of operations for ADT
  - setName, setAddress, setGPA (store data)
  - getName, getAddress, getGPA (retrieve data)
  - print
- Make notes to yourself about any constraints on the data or operations
  - Name must not be blank
  - GPA must be between 0..4

Declaring ADTs

- Most object oriented programming languages have built in support for creating ADTs

- C++ classes contain variable declarations (data) and method prototypes (operations)

- The *public* and *private* tags in class limit access to variables and methods

Declaring ADTs

- C++ classes have *constructor* methods that are called when you declare or create an object
  - these methods are used to initialize the data

- C++ also has *destructor* methods that are automatically called when an object exits scope or is destroyed
  - these methods clean up the data structure

Declaring ADTs

```
class Student
{
public:
   // constructors and destructor
   Student();
   Student(string name, string
     address, float gpa);
   ~Student();

   // get methods
   string getName() const;
   string getAddress() const;
   float getGPA() const;

   // set methods
   void setName(string name);
   void setAddress(string address);
   void setGPA(float gpa);

   // other methods
   void print() const;

private:
   string Name;
   string Address;
   float GPA;
};
```

Using ADTs

- In C++ we can declare objects much like we declare other variables

```
Student john;
Student fred(“Fred”, “Fayetteville”, 4.0);
Student list[10];
```

- These objects are *instances* of the Student class
- We can also use objects inside other class declarations to build more complex ADTs

Using ADTs

- To call methods for an object in C++ we use *object_name.method_name* syntax

```
john.setName("John");
fred.print();
for (int i = 0; i<3; i++)
    list[i].print();
```

- The object before the "." is passed to the method as a *hidden parameter* so we can access the object data within the method

Implementing ADTs

- Implementing the methods in a class is similar to implementing regular functions except:
  - We use “class_name::” before the method name to identify the class this method belongs to
  - We can use all of the variables in a class in any method *without* passing them as parameters
  - We can add “const” after method parameters to say which methods do *not* change any class variables

Implementing ADTs

```
Student::Student()
{
   Name = “ “;
   Address = “ “,
   GPA = 0;
}
```

```
Student::Student(string name, string
     address, float gpa)
{
   Name = name;
   Address = address;
   GPA = gpa;
}

Student::~Student()
{ }
```

Implementing ADTs

```
string Student::getName() const
{ return Name; }

string Student::getAddress() const
{ return Address; }

float Student::getGPA() const
{ return GPA; }
```

```
void Student::setName(string name)
{ Name = name; }

void Student::setAddress(string address)
{ Address = address; }

void Student::setGPA(float gpa)
{ GPA= gpa; }

void Student::print() const
{ cout << “Name:” << Name << “\n“
        << “Address:” << Address << “\n“
        << “GPA:” << GPA << “\n“; }
```

Conclusion

- We will be studying many more ADTs in this class that have different advantages and disadvantages for storing and manipulating data
  - Linked lists
  - Stacks and queues
  - Trees and heaps
  - Hash tables
  - Graphs